Centrifuge Application Framework Software Architecture

Marc Grushcow
NTT SYSTEMS INC
86 Dunblaine Ave , North York, ON M5M 2S1
(416)

NTT SYSTEMS INC
86 Dunblaine Ave , North York, ON M5M 2S1

Project Manager Mr William Fraser
Telephone (416) 635-2160

PWGCS Contract No : W7711-017723
Report No. DRDC Toronto CR 2002-126

On behalf of
Defence R&D Canada

DRDC Toronto Scientific Authority:
Mr William Fraser
Defence R&D Canada - Toronto
1133 Sheppard Avenue W
Toronto, ON, M3M 3B9
416.635.2160

The scientific or technical validity of this Contract Report is entirely the responsibility of the contractor and the contents do not necessarily have the approval or endorsement of Defence R&D Canada

DRDC Toronto

Contract Report
DRDC Toronto CR 2002-126
July 2002

This page intentionally left blank.

Centrifuge Application Framework Software Architecture

Marc Grushcow

NTT SYSTEMS INC
86 Dunblaine Ave.
North York, ON
M5M 2S1

W7711-017723
DRDC Toronto CR 2002-126

On behalf of
Defence R&D Canada

DRDC Toronto

Contract Report
DRDC Toronto CR 2002-126
July 2002

P518022.PDF [Page: 6 of 41]

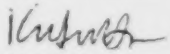
Author

Marc Grushcow

Approved by

Mr. William Fraser
Head, Smart Aircrew Integrated Life Support System

Approved for release by



K.M. Sutton
Chair, Document Review and Library Committee

© Her Majesty the Queen as represented by the Minister of National Defence, 2002

© Sa majesté la reine, représentée par le ministre de la Défense nationale, 2002

Abstract

*NTT SYSTEMS INC.*has developed several applications in support of the human centrifuge facility. These share a common set of basic software building blocks. As a result, all applications present a consistent interface to users and tools designed for one can frequently be integrated into others.

The building blocks are all implemented as *classes* in the Java programming language. Together they constitute a *class library*. The library is partitioned into *packages*, that is, closely related groups of classes. This document provides programmer level documention for this library.

This page intentionally left blank.

PS18022.PDF [Page: 9 of 41]

Executive summary

NTT SYSTEMS INC. has developed a number of applications in support of the human centrifuge facility at DRDRC-Toronto. These are used to design experiments, model G-forces in the gondola, and to analyse physiological and engineering data that is digitized and stored during manned and unmanned runs. All of these applications are used by the same personnel and consequently having a common look and feel is important. Also important from a cost and maintenance perspective is the fact that well designed core components can be shared by all of them.

This document provides programmer level documentation for a Java class library designed for the centrifuge support applications. The design is strongly patterned after the UNIDRAW system, described in *ACM Transactions on Information Systems, July 1990.* The document contains an Introduction followed by one section for each package in the library. Each package's documentation describes its relationship to other packages and identifies its contained classes. Documentation for each class places it in the hierarchy and describes its purpose and operation. UML diagrams are used extensively to show relationships and operational sequences.

The library consists of the following packages:

centrifuge.framework Contains most of the interfaces and classes required to implement the UNIDRAW framework. These include components like the application, manipulator (tool), viewers, and frame proxy.

centrifuge.views This package contains windows that can be subdivided and resized and are containers for other user interface components. They are used to implement multi-document views.

centrifuge.views.chart Classes in this package implement basic graphing components. All applications create their specific data views by subclassing these.

centrifuge.actions Base classes for context menus.

centrifuge.tools Base classes for UNIDRAW style tools (e.g., zoom).

centrifuge.util Utility classes.

centrifute.datasource Classes that provide data to graph views. These can be based on streams or random access files.

P518022 PDF [Page: 10 of 41]

Marc Grushcow; 2002; Centrifuge Application Framework
Software Architecture; — Unassigned —; DRDC Toronto.

Table of contents

Abstract i

Executive summary iii

Table of contents v

List of figures vii

1 Introduction 1

2 The *centrifuge framework* Package 2

2.1 Interfaces 4

2.1.1 App 4

2.1.2 FrameProxyListener 4

2.1.3 Manipulator 5

2.1.4 Viewer 5

2.1.5 ViewerClient 6

2.2 Classes 6

2.2.1 AbstractManipulator 6

2.2.2 BasicApp 6

2.2.3 BasicViewer 6

2.2.4 FrameProxy 7

2.3 Scenarios and Interactions 7

2.3.1 Creating a New Internal or External Window . . . 7

2.3.2 Switching Between Internal and External Frames . 8

2.3.3 Frame Activation 8

2.3.4 Closing Frames 8

2.3.5 BasicViewer Mouse Handling 10

P518022.PDF [Page: 12 of 41]

3 The *centrifuge.views* Package 12
3.1 Classes 13
3.1.1 AbscisaBoundViewer 13
3.1.2 MultiPane 14
3.1.3 MultiPane.Divider 14
3.2 Scenarios and Interactions 15
3.2.1 Adding Graphs 15
4 The *centrifuge.views.chart* Package 15
4.1 Interfaces 17
4.1.1 Graph 17
4.2 Classes 17
4.2.1 AbscisaObservingBasicGraph 17
4.2.2 BasicAxis 18
4.2.3 BasicGraph 19
4.2.4 DataBoundAxis 20
4.2.5 XAxis 20
4.2.6 YAxis 20
5 The *centrifuge.actions* Package 20
5.1 Classes 20
5.1.1 ContextAction 20
6 The *centrifuge.tools* Package 21
6.1 Interfaces 21
6.1.1 Tool 21
6.2 Classes 22
6.2.1 BasicTool 22

— Unassigned —

P518022.PDF [Page: 13 of 41]

7 The *centrifuge.util* Package 22
7.1 Classes 22
7.1.1 GraphInfoStructure 22
8 The *centrifuge.datasource* Package 23
8.1 Interfaces 23
8.1.1 DataSource 23
8.1.2 DataSourceOwner 23
8.1.3 RandomAccessDataSource 24
8.1.4 StaticDataSource 24
8.2 Classes 24
8.2.1 ArrayDataSource 24
8.2.2 DataSourceIndex 24
8.2.3 DataSourceSample 24

List of figures

1 The centrifuge Package 2
2 framework Package Dependencies 3
3 framework and Related Classes 3
4 Creating a New Internal or External Window 7
5 Switching Between Internal and External Frames 9
6 Frame Activation 10
7 BasicViewer Mouse Handling 11
8 views Package Dependencies 12
9 views and Related Classes 13

P518022 PDF [Page: 14 of 41]

10 Adding Graphs 16

11 chart Package Dependencies 17

12 chart and Related Classes 18

1 Introduction

This document describes a set of Java packages that implement a generalized application framework. The packages contain interfaces, abstract classes, and a set of *basic* classes that provide a significant level of functionality It draws heavily on the UNIDRAW system described in *UNIDRAW: A Framework for Building Domain-specific Graphical Editors, Vlissides and Linton, ACM Transactions on Information Systems, Vol 8, No. 3, July 1990*

The framework was developed for a family of interactive applications related to the design and analysis of human centrifuge experiments at DCIEM. These all...

1. *Display graphical (2D and 3D) and other information in a variety of formats in views,*
2. *Allow for user interaction via menus menus and commands;*
3. *Provide variety of tools, some common and some application specific, that the user can wield;*
4. *Allow for user control of screen layout;*
5. *Provide for user extensive customization of UI components including colour and font selection.*

The framework treats all of these components generically. They in turn interact through a set of protocols defined by their interfaces.

Figure 1 shows the relevant packages.

framework *Responsible for menu and toolbar management, frame and window management, and the creation of new viewers.*

tools *Defines the tool manipulation interface and provides a BasicTool class*

actions *Contains classes that can act on view and model components*

util *Various utility classes and interfaces*

datasource *Interfaces and classes for a variety of classes that can provide data to models and views.*

datatrac *Legacy classes, primarily for jar file management, from the IDEA system.*

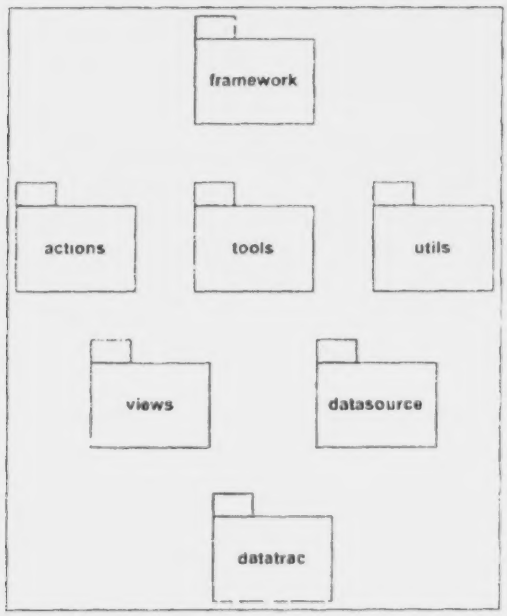


Figure 1: *The centrifuge Package*

2 The centrifuge.framework Package

The framework is responsible for creating an managing the application environment. This includes.

- *Creating a menu and toolbar*
- *Handling user requests to switch from internal to external frames*
- *Creating new views*
- *Managing window activation and deactivation.*
- *Appropriately enabling and disabling tools and menus.*
- *Implementing the tool protocol.*

Figure 2 shows framework's dependencies on other packages.

Figure 3 shows framework and related classes

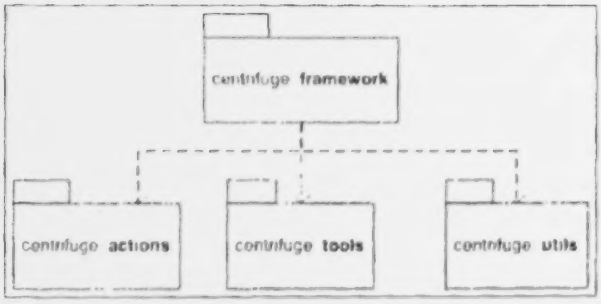


Figure 2: *framework Package Dependencies*

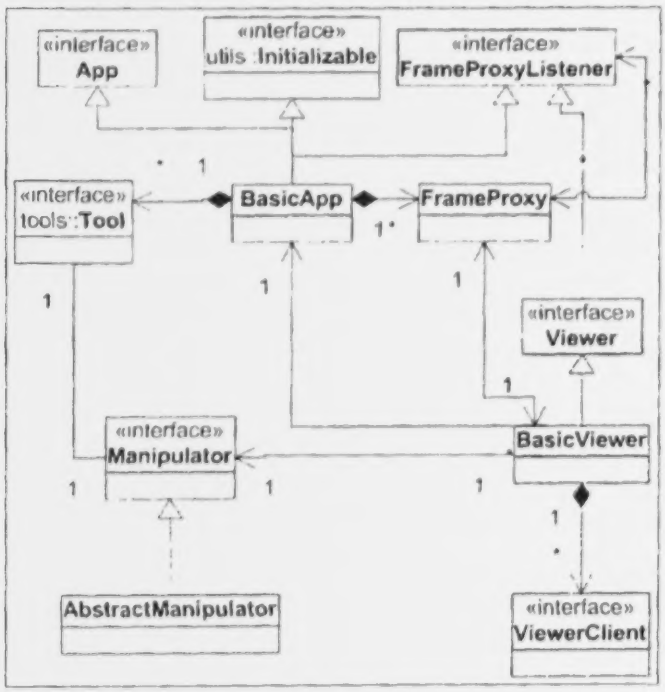


Figure 3: *framework and Related Classes*

App *Defines application level methods for tool protocol and internal/external frame switching*

FrameProxyListener *The listener interface for receiving events from wrapped*

frame

Manipulator *Contains methods for implementing Manipulator design pattern from the Unidraw framework*

Viewer *This interface defines viewer level methods for tool protocol, context menu protocol and menu protocol.*

ViewerClient *This interface contains methods called by Viewer objects on components previously registered with Viewer addClientComponent(Component)*

AbstractManipulator *Contains specific implementation for .isManipulating which currently takes ONLY mouse events and dispatches them to the corresponding javax.swing.event.MouseInputListener methods.*

BasicApp *Contains base structures and methods for providing common functionality on application level for concrete applications.*

BasicViewer *This is a basic implementation of the Viewer interface.*

FrameProxy *This class acts as a proxy, hiding the difference between javax.swing JFrame and javax swing.JInternalFrame.*

2.1 Interfaces

2.1.1 App

Extends:
Implements:

Defines application level methods for tool protocol and internal/external frame switching

2.1.2 FrameProxyListener

Extends:
Implements: *java util EventListener*

The listener interface for receiving events from wrapped frame.

2.1.3 Manipulator

Extends:
Implements: *javax.swing.event.MouseInputListener*

Contains methods for implementing Manipulator design pattern from the Unidraw framework. See:

Unidraw: A Framework for Building
Domain-Specific Graphical Editors

JOHN M. VLISSIDES and MARK A. LINTON
Stanford University

ACM Transaction on Information Systems, Vol. 8, No. 3, July 1990,
Pages 237-268.

The Manipulator protocol is described in Unidraw as follows: "The manipulator protocol is designed to reflect the grasp-wield-effect behavior of tools The Grasp operation takes a window system event (such as a mouse click or key press) and initializes whatever state is needed for the direct manipulation, The Manipulating operation is called repeatedly until the manipulator decides that manipulation has terminated (based on its own termination criteria) and indicates this by returning a false value. The Effect operation gives the manipulator a chance to perform any final actions following the manipulation. Some kinds of direct manipulation may require several submanipulations to progress simultaneously; for instance, the editor may allow the user to manipulate more than one component at a time A manipulator can therefore have children, and the manipulator protocol includes operations for iterating through and manipulating them."

2.1.4 Viewer

Extends:
Implements:

This interface defines viewer level methods for tool protocol, context menu protocol and menu protocol. Viewer maintains a list of ViewerChents. Components from this list will be passed to any tool from application's tool list to triger their enabled/disabled state

2.1.5 ViewerClient

Extends.
Implements:

This interface contains methods called by Viewer objects on components previously registered with Viewer addClientComponent(Component). It is currently used for context menu service.

2.2 Classes

2.2.1 AbstractManipulator

Extends: *javax.swing.JComponent*
Implements: *Manipulator*

Contains specific implementation for isManipulating which currently takes ONLY mouse events and dispatches them to the corresponding javax.swing - event MouseInputListener methods. For any other types of events, currently nothing happens

2.2.2 BasicApp

Extends: *javax.swing.JFrame*
Implements: *App, FrameProxyListener, centrifuge.util Initializable*

Contains base structures and methods for providing common functionality on application level for concrete applications This common functionality includes:

- *internal/external frame switching*
- *handling toolbar and current tool.*
- *handling menu bar and Viewer specific menus*

2.2.3 BasicViewer

Extends: *javax.swing.JPanel*
Implements: *Viewer, java.beans.PropertyChangeListener, FrameProxyListener, centrifuge.util.Initializable*

This is a basic implementation of the Viewer interface. It contains full support for conter menu service. tools service and viewer specifc menu service together with BasicApp

2.2.4 FrameProxy

Extends: *java.lang.Object*
Implements: *javax.swing.RootPaneContainer*

This class acts as a proxy, hiding the difference between javax.swing.JFrame and javax.swing.JInternalFrame. It is used for switching Views back and forth between JFrame and JInternalFrame. It also provides ability for FrameProxyListener to listen uniformly on all events from both types of frames.

2.3 Scenarios and Interactions

2.3.1 Creating a New Internal or External Window

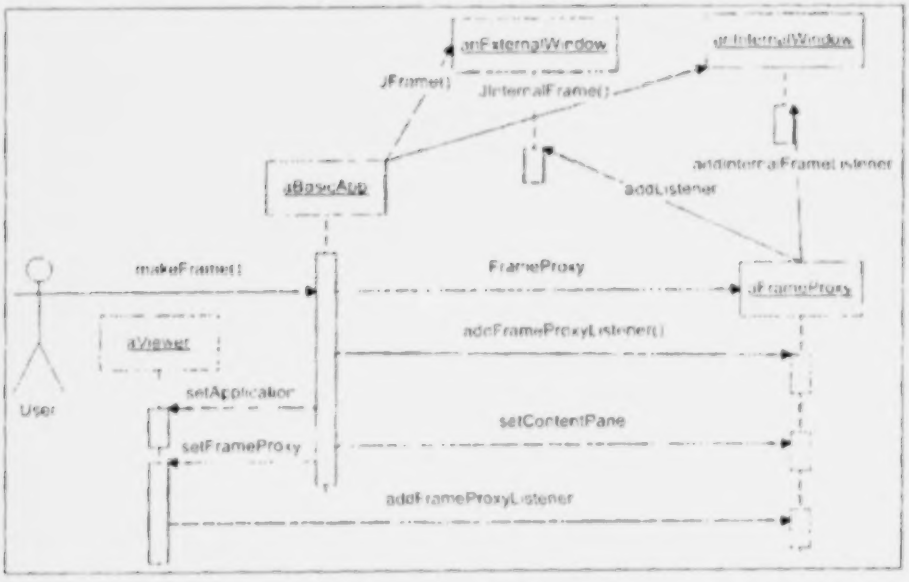


Figure 4: *Creating a New Internal or External Window*

This sequence (Fig 4) is usually triggered by a user who wants a new display window. A kind of Viewer is created by the derived App and given to BasicApp who creates an internal or external frame, depending on the current mode. Note that the actual objects, JFrame and JInternalFrame exhibit different behaviour and do not have a common interface. BasicApp creates a FrameProxy who acts as a facade and presents a single interface to FrameProxyListeners.

BasicApp tells the Viewer and FrameProxy about each other and listens to and retains a reference to the FrameProxy.

2.3.2 Switching Between Internal and External Frames

BasicApp provides for two view display modes: internal frames and external frames. The former is effectively a multi-document interface where all views are contained inside a single window. The switch takes place by removing the View from the existing internal or external frame and placing it in a new external or internal frame. FrameProxy is careful to remove itself as a listener from the original frame before doing this so that it does not get the notification messages arising from content removal and frame destruction. Once the View has been inserted in the new frame, a call to `revalidate` *causes it to be pained. See Fig 5.*

2.3.3 Frame Activation

Each FrameProxy listens to its internal or external frame and receives activation notification (Fig 6). These are different for the different types of frames and are mapped into the common `frameActivated` *notification which is passed on to the FrameProxy listeners. These include BasicApp who manages to tool list. All tools are disabled and the list is passed to the Viewer associated with the now active frame. The sequence diagram is somewhat simplified here. In fact, the Viewer looks at each of its contained clients. If a client is a BasicViewer, the tool list is passed down to it. Otherwise, the client is assumed to be some kind of java.awt.Component. The component is passed to the* `setEnabled` *method of each tool. Tools themselves are responsible for determining whether or not they are prepared to be enabled for any of the components.*

2.3.4 Closing Frames

Since FrameProxy is a WindowListener and InternalFrameListener it detects all closing related events. These are mapped as follows.

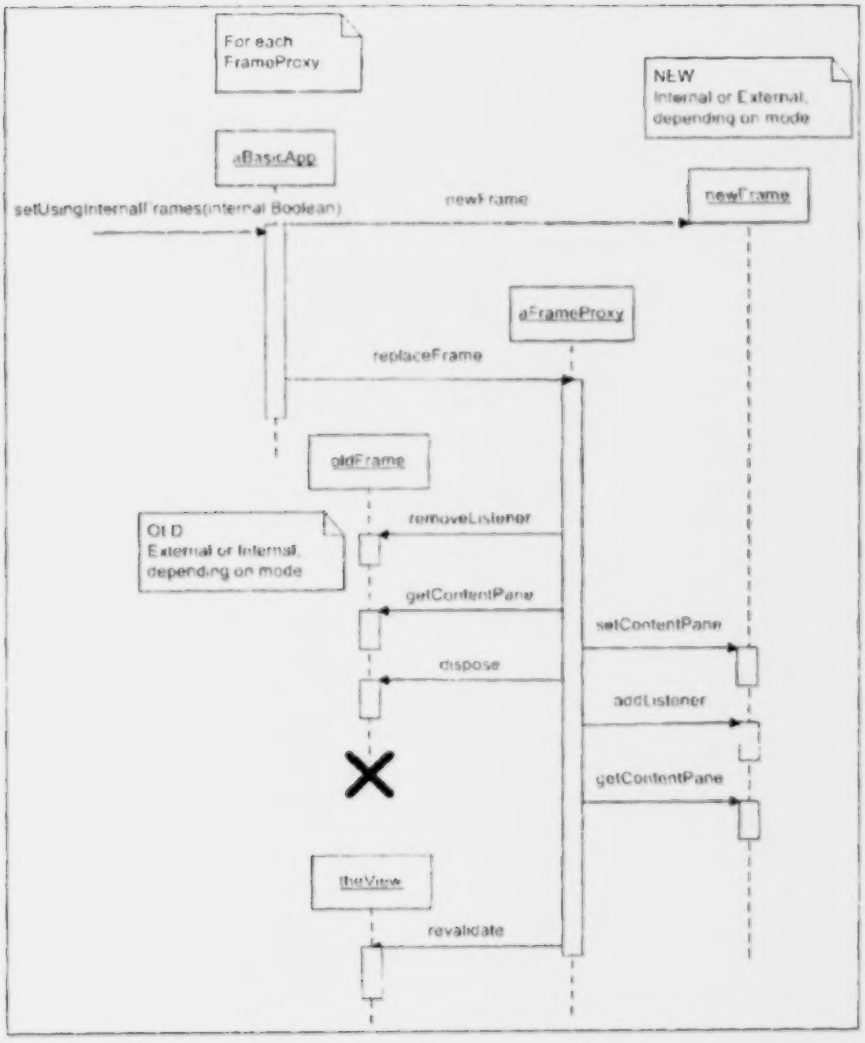


Figure 5: *Switching Between Internal and External Frames*

Raw Event	*Mapped Event*
`windowClosing` `internalFrameClosing`	*frameClosing*
`windowClosed` `internalFrameClosing`	*frameClosed*

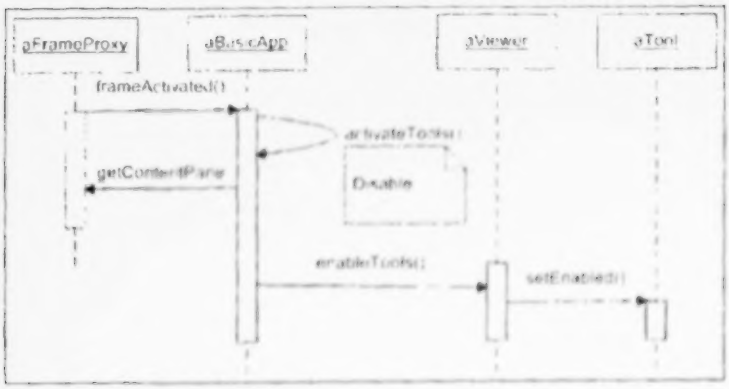


Figure 6: *Frame Activation*

These are in turn handled by FrameProxyListeners. Note that BasicViewer ignores these events, but derived viewers would likely inform their clients and release resources. BasicApp uses `frameClosing` *to remove itself as a listener of the FrameProxy and to remove the FrameProxy from its internal collection.*

2.3.5 BasicViewer Mouse Handling

BasicViewer intercepts and investigates all mouse activity (Fig. 7). Right mouse events are treated as requests to show a context menu. If the event source is an ViewerClient, its context menu is retrieved and each menu item's `setEnabled` *method is called using the event source as a parameter. This allows the menu item to determine whether it can act on the kind of thing under the mouse and if so, whether the source is in the correct state for the action. BasicViewer then displays the popup and the user can activate an action.*

All other mouse events drive the tool manipulation protocol. This can be summarized as follows:

1. *If we do not already have a Manipulator go to the App, get the current Tool and ask it to create a Manipulator. If any of this fails, simply forward the event to the event source. Otherwise, forward the event to the Manipulator's* `start`
2. *Forward the event to the Manipulator's* `isManipulating` *method. Allows it to interact with the event source. Based on the event and state information, the Manipulator decides whether or not it is finished (i.e., has collected whatever information it needs). If so, it returns true, else false.*

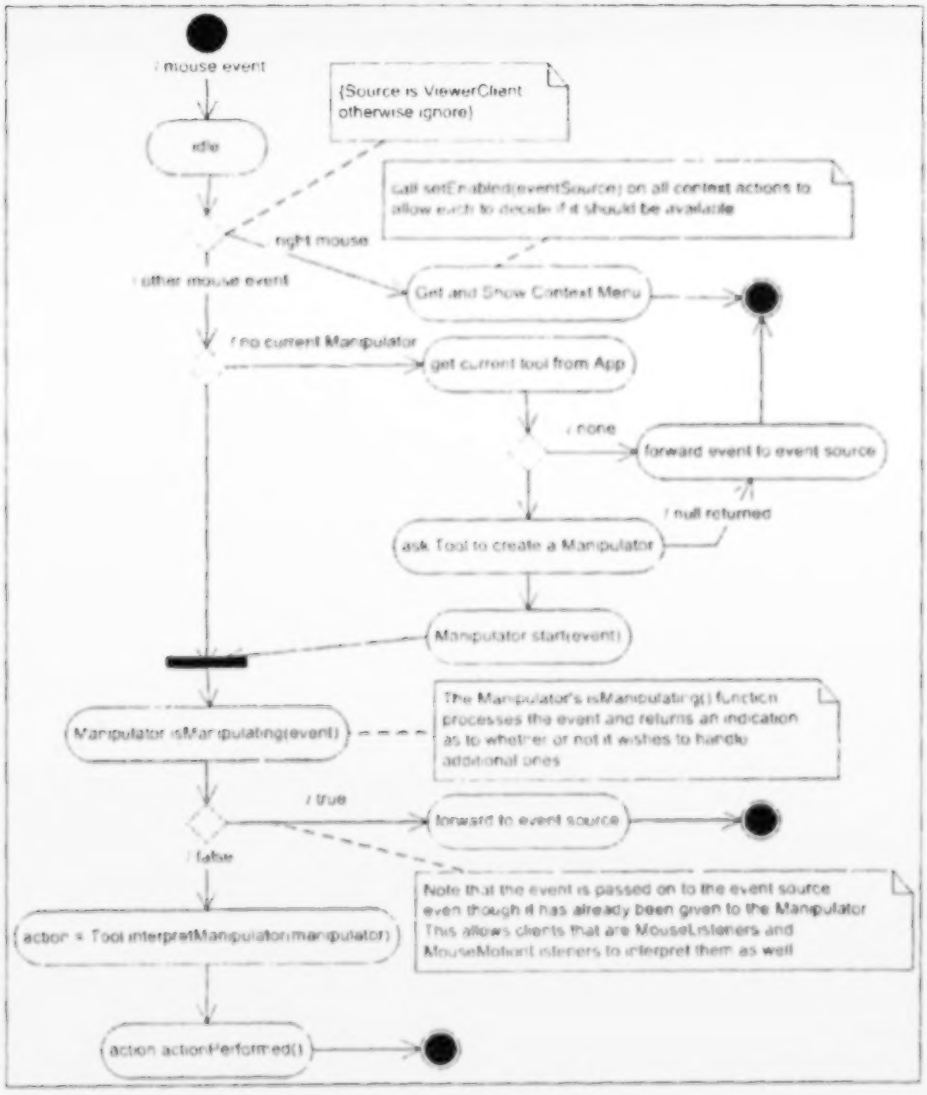

/ mouse event
{Source is ViewerClient
otherwise ignore}
idle
call setEnabled(eventSource) on all context actions to
allow each to decide if it should be available
right mouse
/ other mouse event
Get and Show Context Menu
/ no current Manipulator
get current tool from App
/ none
forward event to event source
/ null returned
ask Tool to create a Manipulator
Manipulator start(event)
Manipulator isManipulating(event)
The Manipulator's isManipulating() function
processes the event and returns an indication
as to whether or not it wishes to handle
additional ones
/ true
forward to event source
/ false
Note that the event is passed on to the event source
even though it has already been given to the Manipulator
This allows clients that are MouseListeners and
MouseMotionListeners to interpret them as well
action = Tool.interpretManipulator(manipulator)
action.actionPerformed()

Figure 7: *BasicViewer Mouse Handling*

3. *If the Manipulator is finished, it is passed back to the current Tool's* `interpretManipulator` *method. The Tool returns an action and the action's* `actionPerformed` *method is called.*

Note that except for when `actionPerformed` *is called, the event is always passed on to the event source if it implements the appropriate listener interface (MouseListener or MouseMotionListener).*

3 The centrifuge.views Package

Viewers are responsible for laying out their internal components and managing interactions with the App, Tools, and Manipulators. Much of this is handled automatically by virtue of their extending BasicViewer. Viewers are application dependent. However, since AbscisaBoundViewer and the classes it depends on are highly generic, they are described here as part of the framework.

Figure 8 shows views' dependencies on other packages

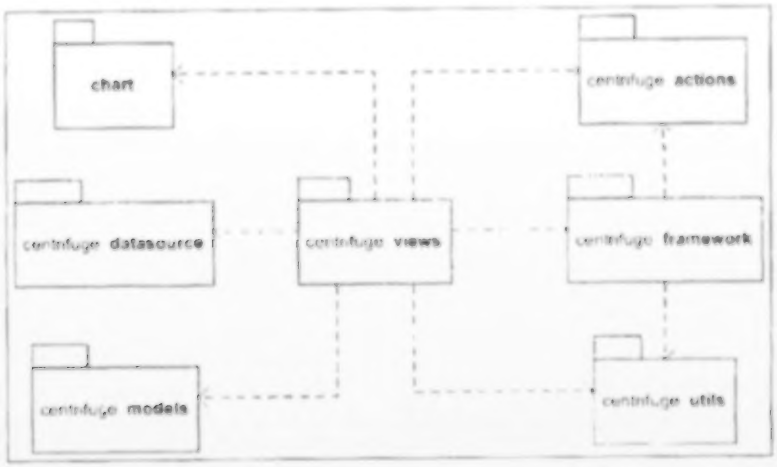


Figure 8: *views Package Dependencies*

Figure 9 shows views and related classes.

AbscisaBoundViewer *This viewer contains a data bound axis (X axis) and multiple graphs inside* `MultiPane`.

MultiPane *This is a simplified version of* `JSplitPane` *which may contain more than two components, separated by a divider component.*

MultiPane.Divider *This is a divider component.*

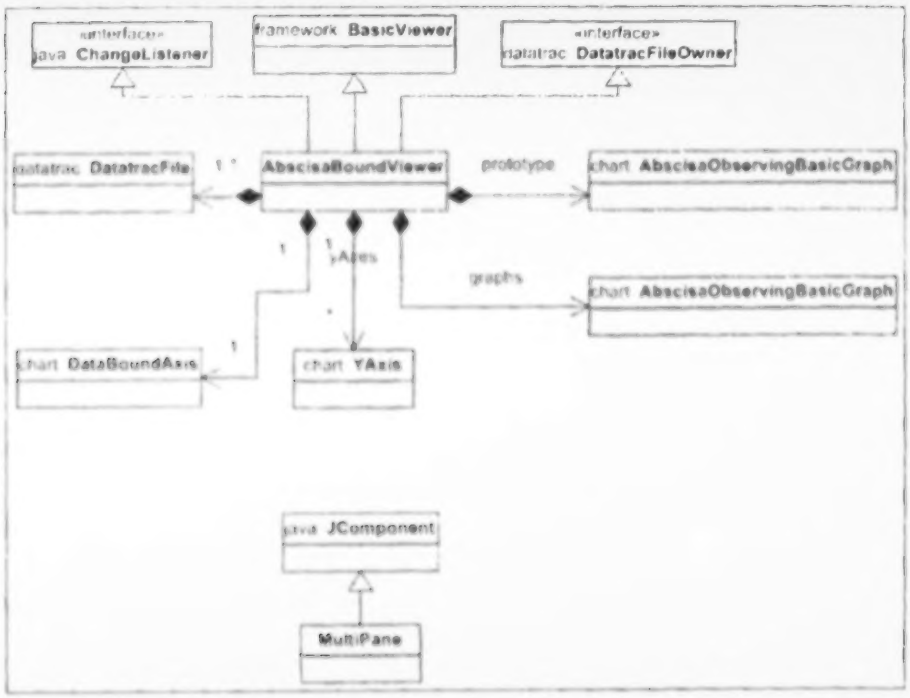


Figure 9: *views and Related Classes*

3.1 Classes

3.1.1 AbscisaBoundViewer

Extends. *centrifuge framework.BasicViewer*
Implements: *centrifuge.datatrac.DatatracFileOwner,*
javax swing.event.ChangeListener

This viewer contains a data bound axis (X axis) and multiple graphs inside `MultiPane`. *All mouse events are handled by its superclass, and this class deals only with managing graphs, setting their data sources and persisting its state to XML stream*

During construction, `AbscisaBoundViewer` *may be initialized with something called a "graph prototype". Graph prototype is a graph which will be added to the pane if user wants to add new graph without providing one. If no proto-*

type was specified and user still wants to add new graph then an instance of `AbscisaObservingBasicGraph` *will be created by default and added to multiple pane.*

`AbscisaBoundViewer` *creates two local actions: one to select a channel and the other to show properties on the Y axis (Persistance model is not completed yet).*

3.1.2 MultiPane

Extends: *javax.swing.JComponent*
Implements:

This is a simplified version of `JSplitPane` *which may contain more than two components, separated by a divider component. Components are managed based on their preferred size which MUST be specified.*

When new components are added to this pane, then if indices are provided they refer only to the indices of 'left' and 'right' components (eg. 0 would refer to the first and 1 to the second component from the outside of this container, although they would be stored internally as 0th, and 2nd component, separated by a divider component which would have index 1) One does not have to worry about divider components as new components are added to this pane - they would be automatically inserted into it. Same way, they would be automatically removed as other components are removed from this pane. However, the existance of dividers as separate components is not completely transparent. Methods such as `getComponentCount()`, `getComponent(int)` *and other methods will count divider components as well.*

NOTE: This is only vertical implementation of multiple split pane

3.1.3 MultiPane.Divider

Extends: *javax.swing.JComponent*
Implements: *javax.swing.event.MouseInputListener*

This is a divider component. It implements mouse input listener in order to change the cursor shape as the mouse enters or exists the component or to

change its position, along with the positions of its left and right components when the mouse is being dragged over it.

3.2 Scenarios and Interactions

3.2.1 Adding Graphs

The AbscisaBoundViewer manages a vertical stack of AbscisaObservingBasicGraphs. It can be passed a graph in the `addNewGraph` *method, or it could have received a cloneable graph as a prototype in its constructor. If it has neither of these, it will create a new AbscisaObservingBasicGraph by default. In the last case, AbscisaBoundViewer will pass the model (the DefaultFloadDataBoundedRangeModel contained in the X-axis to the new graph so that it can add itself as a ChangeListener. The viewer then fakes a PropertyChange event and sends it to the graph causing it to collect the data in the current range.*

The viewer then retrieves the YAxis from the graph and inserts it and the graph into the panel in an East-West arrangement.

Finally, the graph and Y-axis are added to the superclass (BasicViewer) as client components so that they will receive forwarded events and can participate in the tool protocol.

The process is diagrammed in Fig 10

4 The centrifuge.views.chart Package

Figure 11 shows chart' dependencies on other packages.

Figure 12 shows chart and related classes.

Graph *Specifies methods which have to be implemented by different types of chart display components.*

AbscisaObservingBasicGraph *This class is a kind of BasicGraph that listens to DefaultFloatDataBoundedRangeModel and uses their value and extent properties as the start and end of interval displayed along abscisa axis.*

BasicAxis *This class represents a visual representation of an axis*

BasicGraph *This is a basic class to display a chart of samples from any centrifuge.datasource.DataSource object.*

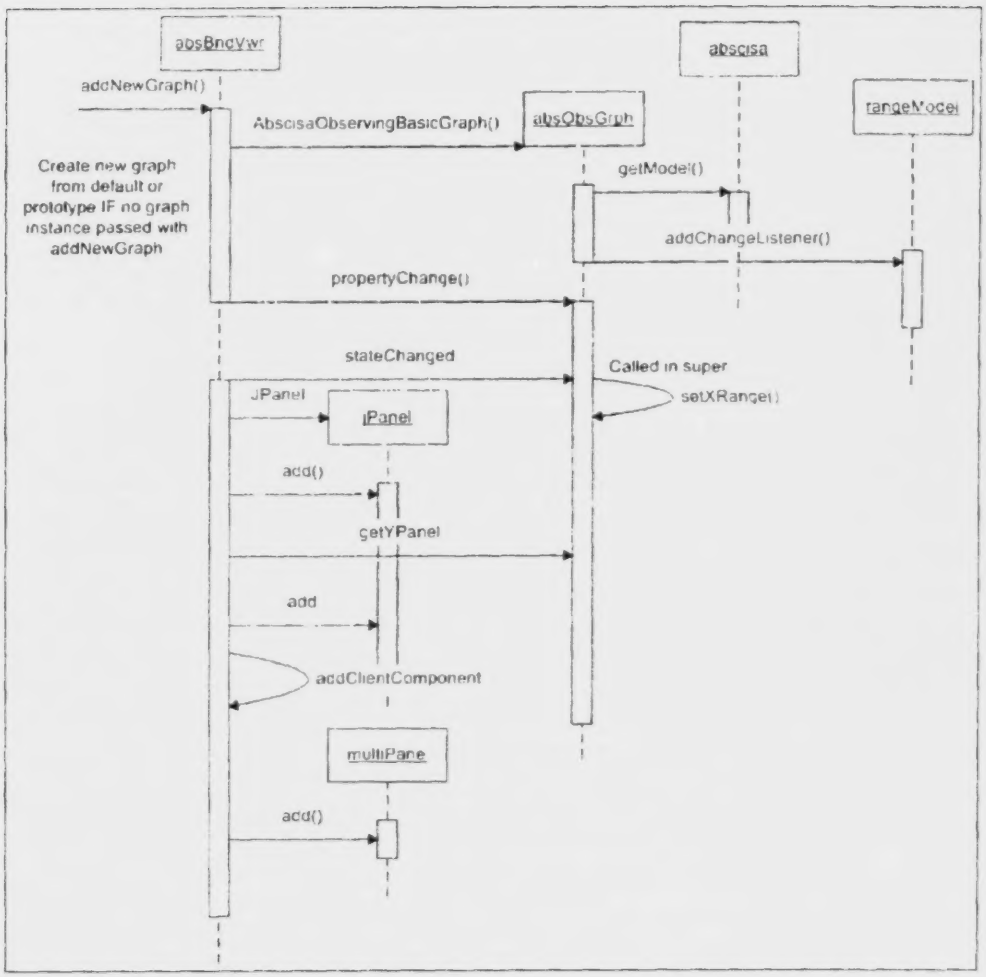


Figure 10: *Adding Graphs*

DataBoundAxis *This axis component contains an instance of* `BasicGraph` *that it draws on its graphics object.*

XAxis *Extends all of its functionality from* `BasicAxis` *and adds its painting and mouse handling routines.*

YAxis *Extends all of its functionality from* `BasicAxis` *and adds its painting and mouse handling routines.*

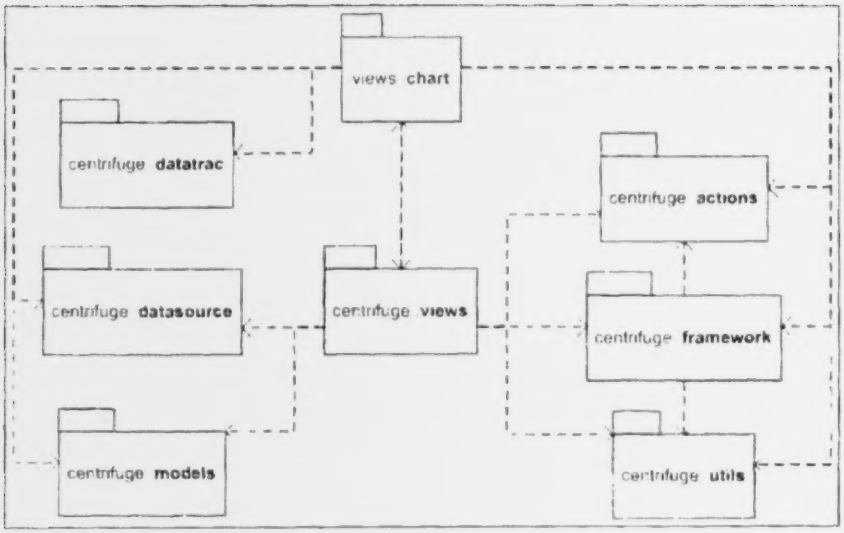


Figure 11: *chart Package Dependencies*

4.1 Interfaces

4.1.1 Graph

Extends.
Implements:

Specifies methods which have to be implemented by different types of chart display components.

4.2 Classes

4.2.1 AbscisaObservingBasicGraph

Extends. *centrifuge.views chart.BasicGraph*
Implements: *javax.swing.event.ChangeListener*

This class is a kind of BasicGraph that listens to DefaultFloatDataBoundedRangeModel and uses their value and extent properties as the start and end of interval displayed along abscisa axis.

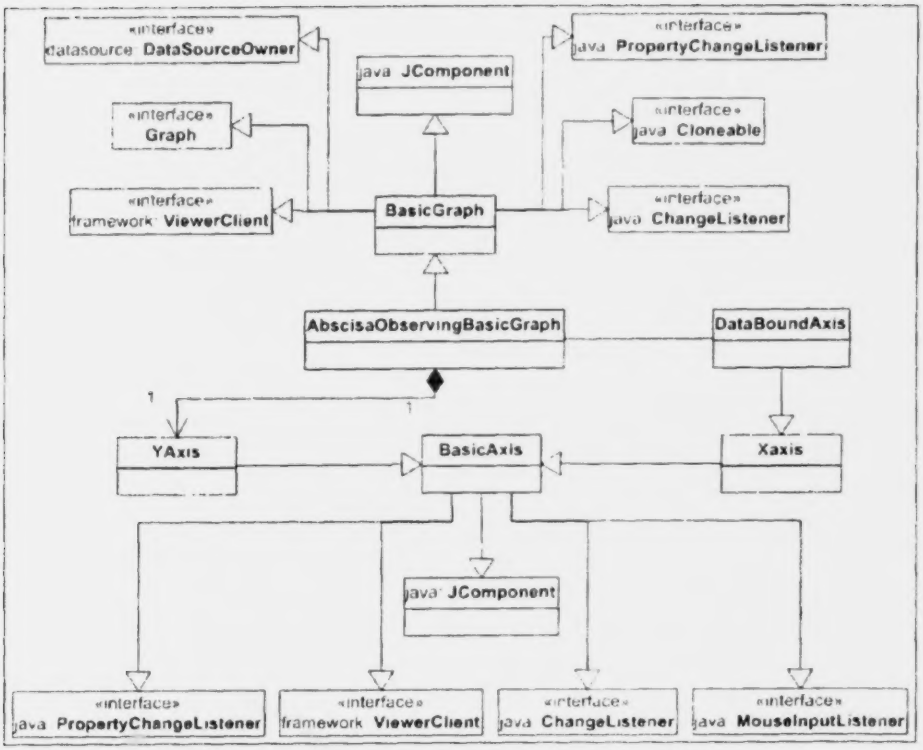


Figure 12: *chart and Related Classes*

4.2.2 BasicAxis

Extends:	*javax.swing.JComponent*
Implements:	*javax.swing.event.ChangeListener,*
	javax.swing.event.MouseInputListener,
	java.beans.PropertyChangeListener,
	centrifuge.framework.ViewerClient,
	centrifuge.util.Initializable

This class represents a visual representation of an axis. It is a view on data bounded range model and contians properties and methods common to both X and Y axis.

Users may interact with it by selcting a part (or range) of the underlying model.

This is called a 'selection rectangle' Its starting point represents the value in the model and its width represents an extent from the model (all translated to graphical values). In addition to these two values, there are three other properties that describe model, and they are. minimum value, maximum value and is-adjusting variable which says if series of changes to the model are in progress. All properties (except the is-adjusting logical variable) can be modified by interacting with the axis view Moving a mouse over the axis will results in different shapes of cursors and they indicate different functionalities available to users There is a short space around the edges of the selection rectangle where cursor will change its shape. That indicates that one may drag the border of the selection rectangle and that way change the value of the underling model. However, if dragging is accomplished with SHIFT key pressed then that would reflect the change of minimum (or maximum) value of the model

This class also implements `Initializable` *interface in order to participate in the persistence protocol It writes its state to an XML writer and is able to reconstructs itself from it at a later time.*

It also conatins two property bags, one with defaulf properties and another one with current properties which may be set localy It is `PropertyChangeListener` *on the parent of its property bag and updates it local properties as notifications arrive from it*

4.2.3 BasicGraph

Extends.	*javax swing.JComponent*
Implements:	*Graph, centrifuge.datasource.DataSourceOwner,*
	javax swing.event.ChangeListener,
	java.beans.PropertyChangeListener,
	centrifuge.framework ViewerClient, java.lang.Cloneable,
	centrifuge util Initializable

This is a basic class to display a chart of samples from any centrifuge.datasource.-DataSource object. It contains a plaggable decimation alhorithm to get a set of data, which visually represents currently selected set of samples from datasource. Currently implemented decimation is designed for fixed rate data sources and is optimazed for speed. It works as follows. Let size.width be chart's area width in pixels.

- *if size.width is greater then number of selected for displaying samples, than all samples will be displayed*

- *if size.width is less then number of selected for displayng samples, than only size width samples will be displayed. This size width samples will be picked by separating whole selected time interval on size.width subintervals and picking the first sample from each subinterval.*

4.2.4 DataBoundAxis

Extends: *centrifuge.views chart.XAxis*
Implements: *centrifuge.datasource.DataSourceOwner*

This axis component contains an instance of `BasicGraph` *that it draws on its graphics object*

4.2.5 XAxis

Extends: *centrifuge views.chart BasicAxis*
Implements:

Extends all of its functionality from `BasicAxis` *and adds its painting and mouse handling routines*

4.2.6 YAxis

Extends: *centrifuge.views chart BasicAxis*
Implements:

Extends all of its functionality from `BasicAxis` *and adds its painting and mouse handling routines.*

5 The centrifuge.actions Package

ContextAction *This is a basic context sensitive, initializable action class*

5.1 Classes

5.1.1 ContextAction

Extends: *javax swing AbstractAction*
Implements.

This is a basic context sensitive, initializable action class. As a part of our application protocol, each action disables/enables itself based on the type of component source, or its state. It may also be initialized with an array of target classes.
As far as the persistence protocol implementation in concearned, this class does most of the work that many actions will need to have done (this is described in methods' descriptions).

6 The centrifuge.tools Package

Tool *Contains methods for implementing Tool design pattern from the Unidraw framework.*

BasicTool *Contains specific implementation for Tool which uses .isOK4Manipulating method both in setEnabled(java.awt Component) and .createManipulator.*

6.1 Interfaces

6.1.1 Tool

Extends:
Implements: *javax swing Action*

Contains methods for implementing Tool design pattern from the Unidraw framework. See:

Unidraw: A Framework for Building
Domain-Specific Graphical Editors

JOHN M. VLISSIDES and MARK A. LINTON
Stanford University

ACM Transaction on Information Systems, Vol. 8, No. 3, July 1990,
Pages 237-268.

The Tool protocol is described in Unidraw as follows: "Conseptually, tools are used on component views as they appear in viewers Whenever a viewer receives an event, it in turn asks the current tool (defined by the enclosing editor object) to produce a manipulator object. A tool implements its CreateManipulator operation to create and initialize an appropriate manipulator, which encapsulates the tools manipulation semantics by defining the three phases (grasp.

weild, effect) of the manipulation. A tool maymodify the contents of the current selection object (also defined by by the enclosing editor) based on the event . A tools InteroretManipulator operation analyzes information in the manipulation that characterizes the manipulation and then creates a command. that carries out the desired effect " In our implementation the command, mentioned above is an Action, returned from .interpretManipulator and it is triggered by centrifuge.framework.BasicViewer But a tool itself is an Action by extending Action, and this, another action is intended to make this tool a current tool when user has pressed tool's button on toolbar The setEnabled(Component) method was added to interface, to allow tool himself decide if he is able to operate on given component.

6.2 Classes

6.2.1 BasicTool

Extends. *centrifuge.actions.ContextAction*
Implements: *Tool*

Contains specific implementation for Tool which uses .isOK4Manipulating method both in setEnabled(java.awt Component) and createManipulator Another implementation specific part is the way in which tool becomes the current tool This class fires propertyChange events for fictitious property named SELECTED. centrifuge.framework.BasicApp listens for this property and performs selection of the current tool.

7 The centrifuge.util Package

GraphInfoStructure *This class represents a structure that describes one particular channel.*

7.1 Classes

7.1.1 GraphInfoStructure

Extends: *java.lang.Object*
Implements:

This class represents a structure that describes one particular channel. It contains the following information: channel description, channel ID, minimum

and maximum value

8 The centrifuge.datasource Package

DataSource *This is a basic interface for all kind of data sources.*

DataSourceOwner *Defines getter and setter methods for data source owners.*

RandomAccessDataSource *This is a basic interface for a data source which provides random access to its samples*

StaticDataSource *This is a basic interface for a persistent data source (such as disk file).*

ArrayDataSource *Implementation of RandomAccessDataSource backed by an array.*

DataSourceIndex *This class conatins the information (index and x value) of the current sample*

DataSourceSample *Wrapper for the array of samples*

8.1 Interfaces

8.1.1 DataSource

Extends:
Implements.

This is a basic interface for all kind of data sources

8.1.2 DataSourceOwner

Extends.
Implements:

Defines getter and setter methods for data source owners.

8.1.3 RandomAccessDataSource

Extends:
Implements: *StaticDataSource*

This is a basic interface for a data source which provides random access to its samples

8.1.4 StaticDataSource

Extends:
Implements: *DataSource*

This is a basic interface for a persistent data source (such as disk file)

8.2 Classes

8.2.1 ArrayDataSource

Extends: *java.lang Object*
Implements: *RandomAccessDataSource*

Implementation of RandomAccessDataSource backed by an array.

8.2.2 DataSourceIndex

Extends: *java.lang Object*
Implements: *java lang.Cloneable*

This class conatins the information (index and x value) of the current sample. Tells the caller how far into the sample stream we are. Does not assume periodic sampling (ie., no assumed time base) Tells the caller where the last sample was in the sample stream

8.2.3 DataSourceSample

Extends. *java lang.Object*
Implements: *java.lang.Cloneable*

Wrapper for the array of samples. Samples are described by two arrays of the same size, each representing a series of x and y values. A sample is a pair of

elements from x and y arrays with the same index. For example, 5th sample of the array x could be accessed as

```
DataSourceSample dss;
... // Somewhere dss was created and filled in
double sampleX = dss.x[4];
```

DOCUMENT CONTROL DATA SHEET

1a. PERFORMING AGENCY NTT SYSTEMS INC ,86 Dunblaine Ave , North York, ON M5M 2S1	2 SECURITY CLASSIFICATION UNCLASSIFIED Unlimited distribution -
1b PUBLISHING AGENCY DRDC Toronto	
3 TITLE (U) Centrifuge Application Framework Software Architecture	
4 AUTHORS Marc Grushcow	
5 DATE OF PUBLICATION July 8 , 2002	6 NO OF PAGES 32
7 DESCRIPTIVE NOTES	
8 SPONSORING/MONITORING/CONTRACTING/TASKING AGENCY Sponsoring Agency Monitoring Agency Contracting Agency DRDC Toronto Tasking Agency	
9 ORIGINATORS DOCUMENT NO Contract Report CR 2002-126	10 CONTRACT GRANT AND/OR PROJECT NO W7711-017723
12 DOCUMENT RELEASABILITY Unlimited distribution	
13 DOCUMENT ANNOUNCEMENT Unlimited announcement	

14 ABSTRACT

(U) NTT SYSTEMS INC. has developed several applications in support of the human centrifuge facility. These share a common set of basic software building blocks. As a result, all applications present a consistent interface to users and tools designed for one can frequently be integrated into others.

The building blocks are all implemented as classes in the Java programming language. Together they constitute a class library. The library is partitioned into packages, that is, closely related groups of classes. This document provides programmer level documentation for this library.

(U)

15 KEYWORDS, DESCRIPTORS or IDENTIFIERS

(U) centrifuge; software

#518022
CA021300